ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

Revisiting Edge Pooling in Graph Neural Networks

Francesco Landolfi*

Università di Pisa - Department of Computer Science
Largo Bruno Pontecorvo, 3, 56127, Pisa - Italy

Abstract. Sparse pooling methods for graph neural networks typically perform graph reduction by keeping only the top-k vertices according to an adaptive scoring function. Although fast and scalable, these methods destroy the relational information of the graph and possibly make it disconnected. EdgePool is one of the few sparse alternatives that preserve the connectivity of the input graph by performing a series of edge contractions according to an adaptive scoring of the edges, but it has the drawback of being sequential and not scalable on large scale graphs. In this paper we show that EdgePool can be efficiently computed adapting a well-known parallel algorithm from literature, and we also propose a novel, parallel alternative that leverages on an adaptive scoring function of the nodes. We test both methods on standard benchmark datasets, showing that they generally outperform other sparse pooling methods from the literature.

1 Introduction

Graph neural networks (GNNs) [1] can be designed to reduce the input graphs by means of pooling mechanisms, generally placed after one or more layers of graph convolution. The irregularity of graphs does not allow for a trivial definition of down-sampling as the one used for, e.g., audio signals, images, or voxel data. This has led to a multitude of different definitions of pooling for GNNs, that can be roughly divided into two classes: *dense* and *sparse* methods. Dense methods, like DiffPool [25] or MinCutPool [2], adaptively compute soft-assignments of the vertices to a fixed number of clusters, which is in general defined as a fraction of the average graph size in the training set. They require up to $O(rn^2)$ space to generate the soft-assignment matrix, with $r \in (0, 1)$ and n is the number of vertices, which can be prohibitively high for large scale graphs. Sparse methods, instead, are pooling mechanisms that are able to reduce the graph in linear space w.r.t. the number of vertices or edges. Among them, there are vertex selection methods, such as TopKPool [11], SAGPool [15], ASAPool [19], and PANPool [17], that adaptively assign a score to every vertex, using different kinds of neural networks depending on the model, and then keep only the top-k ones, with $k = \lceil rn \rceil$, and drop all the other vertices. Although fast and scalable, this kind of methods may drop, along with the vertices, important relational information and connectivity properties. EdgePool [8] is a sparse method that does not have these drawbacks, as it performs graph reduction by iteratively contracting the edges, following an ordering defined by adaptive scoring functions, unless they are incident to an already contracted edge, until no other edge is contractable. Although not stated in the paper, this method is equivalent to finding and contracting a *maximal matching* of the graph, which is a common coarsening step already adopted in multi-level graph partitioning [13], spectral clustering [7], spectral reduction [16], and graph

*I would like to thank Davide Bacciu and Alessio Conte for their useful suggestions.

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

drawing [22]. In this paper we show how the well-known maximal matching algorithm of Blelloch et al. [3] can be exploited to parallelize both EdgePool and a novel vertex-scoring variant that acts as a intermediate step between vertex-selection methods and EdgePool.

2 Edge pooling via parallel maximal matching

Notation and preliminaries. We represent a graph with $G = (V, E)$, where $V = \{1, \dots, n\}$ and $E \subseteq V \times V$ are, respectively, the vertex and the edge sets of G, with $|E| = m$. Given $U \subseteq V$, we denote with $G[U]$ the subgraph of G induced by the vertices in U. We represent with $\mathbf{A} \in \mathbb{R}^{n \times n}$ the (symmetric) adjacency matrix of a graph and with $\mathbf{X} \in \mathbb{R}^{n \times f}$ its feature matrix, whose rows represent the feature vectors of the vertices. We denote with $N(v)$ the neighbors of $v \in V$ ($N[v]$ if inclusive) and, with a slight abuse of notation, we will also denote with $N(e)$ the neighbors of an edge $e = uv \in E$, that is, the set of edges incident to u or v. A *matching* $M \subseteq E$ is a set of edges such that any two edges in M share no vertices. A matching is *maximal* if any edge in $E \setminus M$ has a neighboring edge in M. *Contracting* an edge $uv \in E$ means replacing u and v with a single vertex incident to all the edges in $N(uv)$. A *ranking* is an injective function $\pi : S \to \mathbb{N}$ mapping any element in the set S to its position among the other elements with respect to a given ordering.

Ranking the edges by their scores. Diehl et al. [8] proposed an edge contraction pooling method, EdgePool, which works as follows: for every edge $uv \in E$, compute its *score*, $\vec{s}_{uv} = \sigma(\mathbf{w}^\top [\mathbf{x}_u, \mathbf{x}_v] + b)$, where $\mathbf{w} \in \mathbb{R}^{2f}$ and $b \in \mathbb{R}$ are parameters of the scoring function, and σ is an activation function, which the authors set to *tanh* or *softmax*, the latter evaluated over every vertex's neighborhood. Then, the graph is reduced by iteratively contracting the edge with highest score whose vertices have not already been contracted. The feature vectors of the merged vertices is then obtained with $\mathbf{x}_{uv} = \vec{s}_{uv} \cdot (\mathbf{x}_u + \mathbf{x}_v)$. Notice that weighting (or *gating*) the feature vectors by the edge's score is necessary to make the scoring function end-to-end differentiable.

The process of greedily inserting an edge in the matching following a *fixed* ordering is known as *lexicographically-first* maximal matching [6], a problem for which Blelloch et al. [3] proposed a parallel algorithm, that we restate in Algorithm 1. Since the scores do not change after the selection of an edge, we can parallelize the computation of the set of edges extracted by EdgePool by means of this algorithm. To do this, we first have to make the edge scores invariant to their orientation, as Algorithm 1 assumes the graph to be undirected. We obtain this by computing the *undirected score* as $s_{uv} = \max(\vec{s}_{uv}, \vec{s}_{vu})$. (Clearly, the score of an edge selected by EdgePool is equal to its undirected score, hence this modification as no impact on the final result.) Finally, we compute the *ranking* of the edges based on their undirected score, by sorting them in parallel in *non-increasing* order of s.

Ranking the edges by the score of their vertices. The previous formulation has the drawback of using the score as a gating function only for the matched edges, while unmatched ones will bypass both the reduction and the scoring function. This may cause a form of "survivorship bias" in the selection phase: rightfully selected edges (i.e., with a high score) will produce a low gradient, hence a marginal correction of the scoring function, while wrongfully excluded ones (i.e., edges that should have deserved a higher score) will be detached from the computation graph, causing their score not

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

Algorithm 1 Parallel greedy maximal matching (MM) algorithm, from Blelloch et al. [3]. Given a graph G and an edge ranking π, returns a maximal matching in G.

1: **function** MM($G = (V, E)$, $\pi : E \to \mathbb{N}$)
2: **if** $|E| = 0$ **then return** $\emptyset$
3: $S \leftarrow \{e \in E \mid \forall f \in N(e).\ \pi(e) < \pi(f)\}$
4: $R \leftarrow V \setminus \bigcup_{uv \in S} \{u, v\}$
5: **return** $S \cup \text{MM}(G[R], \pi)$

Algorithm 2 Parallel greedy vertex-ordered maximal matching (VOMM) algorithm. Given a graph G and a vertex ranking π, returns a maximal matching in G.

1: **function** VOMM($G = (V, E)$, $\pi : V \to \mathbb{N}$)
2: $\hat{\pi} \leftarrow \max_{v \in V} \pi(v)$
3: Define $\pi' : E \to \mathbb{N}$ as $\pi'(uv) = \min(\hat{\pi} \cdot \pi(u) + \pi(v),\ \hat{\pi} \cdot \pi(v) + \pi(u))$
4: **return** MM(G, π')

being corrected. Here we propose a variation of EdgePool that computes a score for every vertex, that will also gate its feature vector regardless of the matching outcome. For every vertex $v \in V$, its score is computed as $s_v = \sigma(\mathbf{w}^\top \mathbf{x}_v + b)$, where $\mathbf{w} \in \mathbb{R}^f$ and $b \in \mathbb{R}$ are parameters of the scoring function, and σ is the *sigmoid* activation function. All features are gated by the score of their vertices, i.e., $\mathbf{x}'_v = s_v \mathbf{x}_v$, while features of matched edges are also summed together, i.e., $\mathbf{x}'_{uv} = s_u \mathbf{x}_u + s_v \mathbf{x}_v$. This reduction can also be expressed in matrix notation as $\mathbf{X}' = \mathbf{P}\mathbf{X}$, where $\mathbf{P} \in \mathbb{R}^{c \times n}$ is a partition matrix having as only non-zero element $\mathbf{P}_{\kappa_u u} = s_u$, where κ_u is the index assigned to the cluster of u, which is either a singleton cluster (u has not been matched) or a two-elements cluster (u is matched with an adjacent vertex). The coarsened graph is obtained following the reduction scheme typical of Nyström methods [16], i.e., $\mathbf{A}' = \mathbf{P}^{+\top}\mathbf{A}\mathbf{P}^{+}$, where $\mathbf{P}^{+}$ denotes the Moore-Penrose pseudo-inverse of $\mathbf{P}$, and $\mathbf{P}^{+\top}$ the transpose pseudo-inverse (notice that, being a partition matrix, $\mathbf{P}$ is easily pseudo-invertible [see 16, Proposition 6]).

Regarding the matching algorithm, in our EdgePool variation we will iteratively select the edges in a way to *minimize* the total score of the matched vertices (or, equivalently, to maximize $1 - s$). This will cause low scoring vertices to be merged together, while leaving the more salient ones possibly unmatched, thus mimicking the selection procedure common in top-k pooling methods [11, 15, 19, 17].

The problem of finding the matching maximizing a positive weight on the vertices (or, in our case, a score) is known as *maximum vertex-weight matching* (MVM) [21]. In Algorithm 2 we propose a parallel lexicographically-first variant of the greedy MVM algorithm of Halappanavar [12, HybridHalf], that we use to compute the matching by ranking the vertices in *non-decreasing* order of s.

Theoretical and complexity analysis. By Lemma 5.3 of [3], we know that, for a random ordering on the edges, Algorithm 1 requires $O(m)$ work and $O(\log^3 n)$ depth with high probability. Given a random scoring of the edges, we can compute their ranking, needed by Algorithm 1, in $O(m \log n)$ work and $O(\log n)$ depth using, e.g., parallel quick-sort [4]. Hence, the computation of the matching needed by EdgePool is indeed parallel on average. The complexity of Algorithm 2 is a bit more involved, since the edge ranking computed in Line 3 of Algorithm 2 is no longer random (e.g., low ranking edges will be connected to the low ranking vertices). Nevertheless, we can show that its depth complexity is still poly-logarithmic on average by leveraging on the theoretical

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

framework of Blelloch et al. [3] (the formal proof is omitted for lack of space).

Theorem 1. *For a random ordering on the vertices, Algorithm 2 can be implemented to run in $O(m)$ total work and $O(\log^3 n)$ depth w.h.p.*

Regarding the matching quality, we can also show that Algorithm 2 becomes equivalent to the sequential MVM algorithm of Halappanavar [12, HybridHalf] when the vertices are ranked in non-increasing ordering of a given weighting function $w : V \to \mathbb{R}_+$. Hence, the analysis of Halappanavar [12] also holds for Algorithm 2 and we have the following proposition.

Corollary 2. *Let $G = (V, E)$ be a graph and $\pi : V \to \mathbb{N}$ be the ranking of its vertices in non-decreasing order of a given weighting function $w : V \to \mathbb{R}_+$. Then, Algorithm 2 will compute a $\frac{1}{2}$-approximation to a maximum vertex-weight matching in G.*

3 Experiments

We tested both EdgePool and our variation, referred to as EdgePoolV2 hereafter, on DD [9], GITHUB [20], REDDIT-B and -5K/12K [24]. All datasets were divided in *training* (70%), *validation* (10%), and *test* (20%) sets using a (fixed) stratified split. In Table 1 we show the average accuracy obtained by different pooling methods on the selected datasets. All models have the same architecture: 3 GNN layers, optionally interleaved by a pooling layer, followed by a global *max/sum* pooling, and a final 2-layer MLP. For every pooling configuration we performed a grid-search model selection on the validation set, with a base search space common to all methods (learning rate in $\{10^{-3}, 10^{-4}\}$, hidden units and batch size in {32, 64, 128}, and GNN layer in {GCN [14], GIN [23], and GATv2 [5]}), extended with a space specific to the pooling methods (reduction ration in {0.5, 0.2, 0.1}, except for EdgePool and EdgePoolV2, that *required no hyper-parameter tuning*). For PANPool we fixed the GNN layer to PANConv [17], which was required by the pooling layer, and added instead the path length hyper parameter in the search space ({1, 2, 3}). The results in Table 1 are obtained using the best configuration found, and averaged among 10 runs of retraining (on the training set, with early stopping on the validation set) and prediction (on the test set) with 10 different random initializations of the model parameters. All the models were implemented using the PyG library [10], where all pooling layers were readily available. We implemented EdgePool and EdgePoolV2 to allow their computation to run entirely on GPU, using the parallel algorithms defined in Algorithms 1 and 2 and discussed in Section 2. On a side note, we report that our implementation is not only dramatically faster[1] than the one already available on PyG when running on GPU (averagely 60× faster in computing a batch of 128 graphs from REDDIT-12K), but also sensibly faster on CPU (resp. 5× faster).

We can see from the results in Table 1 that EdgePool and EdgePoolV2 perform almost on par on all the selected datasets, with results within their respective standard deviation ranges, with the exception of REDDIT-12K, where EdgePool perform significantly better than EdgePoolV2. It is evident instead how edge contraction methods are superior with respect to vertex selection variants, with an increment of more than 5 points of accuracy on REDDIT datasets. We can also see that the baseline

[1] On a machine with a Intel i7-4790K CPU, 32GB of RAM, and a Nvidia GTX1060 GPU, with 6GB of on-board memory.

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

Pooling	DD	RDT-B	RDT-5K	RDT-12K	GITHUB
None	75.51 ± 1.07	78.40 ± 8.68	48.32 ± 2.38	45.04 ± 6.63	$\mathbf{69.89 \pm 0.28}$
Graclus [7]	75.17 ± 2.11	84.05 ± 5.81	43.22 ± 12.24	43.08 ± 9.32	67.64 ± 0.57
TopKPool [11]	74.92 ± 2.03	81.10 ± 3.82	45.28 ± 3.88	38.55 ± 2.35	65.93 ± 0.45
SAGPool [15]	73.26 ± 2.26	84.90 ± 3.94	46.29 ± 5.61	42.30 ± 3.70	64.29 ± 5.70
ASAPool [19]	73.73 ± 2.18	78.37 ± 5.22	39.53 ± 7.76	39.14 ± 3.58	66.98 ± 0.96
PANPool [17]	73.26 ± 1.94	77.44 ± 4.95	46.04 ± 3.78	40.97 ± 3.02	62.48 ± 2.84
EdgePool (*tanh*)	75.42 ± 1.41	$\mathbf{87.33 \pm 1.28}$	$\mathbf{52.39 \pm 1.11}$	$\mathbf{47.20 \pm 0.91}$	63.20 ± 4.76
EdgePool (*softmax*)	75.13 ± 2.07	86.53 ± 1.25	40.84 ± 8.32	43.08 ± 7.18	$\boldsymbol{69.17 \pm 0.73}$
EdgePoolV2	$\mathbf{77.29 \pm 1.63}$	$\boldsymbol{86.80 \pm 6.16}$	$\boldsymbol{52.27 \pm 1.21}$	$\boldsymbol{46.30 \pm 0.95}$	67.25 ± 0.58
EdgePoolV2 ($\sigma \circ \mathcal{N}$)	75.04 ± 1.62	84.55 ± 2.60	49.28 ± 1.85	45.43 ± 6.69	66.88 ± 4.75
EdgePoolV2 ($\mathcal{U}$)	$\boldsymbol{75.59 \pm 2.15}$	81.97 ± 3.66	43.60 ± 7.47	45.82 ± 0.87	66.54 ± 2.78

Table 1: Classification accuracy *(mean ± std)* in standard benchmark datasets. Values in bold/italic highlight the best two results on each dataset.

with no pooling layers performs generally better than vertex selection based methods but sensibly worse than EdgePool and EdgePoolV2, except on GITHUB, where it obtained the best result. Following the experimental evaluation of Mesquita et al. [18], we also compared our method with other two baselines that generate random vertices scores using a uniform distribution ($\mathcal{U}(0,1)$) and a logit-normal distribution ($\sigma(\mathcal{N}(0,1))$). These baselines obtained a comparable result only on the smallest dataset (DD), but showed not to be as effective as the one using a learned scoring function. Still, both on GITHUB and DD the difference in terms of average accuracy is marginal, proving that edge-based coarsening methods are powerful and reliable in general.

4 Conclusions

We presented a parallel implementation of EdgePool that leverages on well-known reduction techniques borrowed from graph theory, allowing its deployment on graph machine learning pipelines for fast end-to-end training and prediction on GPU. We also presented a novel variant that, differently from the former method, contracts the edges attached to negligible vertices according to a learned scoring function, following the same intuition of top-k methods for graph pooling but, at the same time, keeping the graph connected, thus avoiding one of their main disadvantages. Finally, we empirically showed that edge-contraction methods are generally superior to vertex-selection ones on several graph classification tasks, thus proving them to be a strong and reliable choice for graph pooling.

References

[1] D. Bacciu, F. Errica, A. Micheli, and M. Podda. "A gentle introduction to deep learning for graphs". *Neural Networks* 129 (2020).

[2] F. M. Bianchi, D. Grattarola, and C. Alippi. "Spectral Clustering with Graph Neural Networks for Graph Pooling". *International Conference on Machine Learning* 1 (2020).

[3] G. E. Blelloch, J. T. Fineman, and J. Shun. "Greedy sequential maximal independent set and matching are parallel on average". *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*. 2012.

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

[4] G. E. Blelloch and B. M. Maggs. "Parallel algorithms". *Algorithms and theory of computation handbook: special topics and techniques*. 2nd ed. 2010.

[5] S. Brody, U. Alon, and E. Yahav. "How Attentive are Graph Attention Networks?" *International Conference on Learning Representations*. 2021.

[6] S. A. Cook. "A taxonomy of problems with fast parallel algorithms". *Information and Control* 64.1 (1985).

[7] I. S. Dhillon, Y. Guan, and B. Kulis. "Weighted Graph Cuts without Eigenvectors A Multilevel Approach". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.11 (2007).

[8] F. Diehl, T. Brunner, M. T. Le, and A. Knoll. "Towards Graph Pooling by Edge Contraction". *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data*. 2019.

[9] P. D. Dobson and A. J. Doig. "Distinguishing enzyme structures from non-enzymes without alignments". *Journal of Molecular Biology* 330.4 (2003).

[10] M. Fey and J. E. Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.

[11] H. Gao and S. Ji. "Graph U-Nets". *International Conference on Machine Learning*. 2019.

[12] M. Halappanavar. "Algorithms for Vertex-Weighted Matching in Graphs". *Computer Science Theses & Dissertations* (2009).

[13] G. Karypis and V. Kumar. "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs". *SIAM Journal on Scientific Computing* 20.1 (1998).

[14] T. N. Kipf and M. Welling. "Semi-Supervised Classification with Graph Convolutional Networks". *International Conference on Learning Representations*. 2017.

[15] J. Lee, I. Lee, and J. Kang. "Self-Attention Graph Pooling". *International Conference on Machine Learning*. 2019.

[16] A. Loukas. "Graph Reduction with Spectral and Cut Guarantees". *Journal of Machine Learning Research* 20.116 (2019).

[17] Z. Ma, J. Xuan, Y. G. Wang, M. Li, and P. Lio. "Path Integral Based Convolution and Pooling for Graph Neural Networks". *Advances in Neural Information Processing Systems (NeurIPS) 33*. 2020.

[18] D. Mesquita, A. H. Souza, and S. Kaski. "Rethinking pooling in graph neural networks". *Advances in Neural Information Processing Systems (NeurIPS) 33*. 2020.

[19] E. Ranjan, S. Sanyal, and P. Talukdar. "ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations". *Proceedings of the AAAI Conference on Artificial Intelligence* 34.04 (2020).

[20] B. Rozemberczki, O. Kiss, and R. Sarkar. "Karate Club: An API Oriented Open-Source Python Framework for Unsupervised Learning on Graphs". *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020.

[21] T. H. Spencer and E. W. Mayr. "Node weighted matching". *Automata, Languages and Programming*. 1984.

[22] C. Walshaw. "A Multilevel Algorithm for Force-Directed Graph-Drawing". *Graph Algorithms and Applications 4*. 2006.

[23] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. "How Powerful are Graph Neural Networks?" *International Conference on Learning Representations*. 2019.

[24] P. Yanardag and S. Vishwanathan. "Deep Graph Kernels". *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2015.

[25] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. "Hierarchical Graph Representation Learning with Differentiable Pooling". *Advances in Neural Information Processing Systems*. Vol. 31. 2018.